Spring MVC 4 RESTFul Web Services CRUD例子（带源码）【这才是restful，超经典】

 2016年05月07日 18:57:35

标签：Spring MVC 4 / RESTFul Web Services / CRUD例子 / 这才是restful 32060

本系列其他教程正在陆续翻译中，点击分类：spring 4 mvc 进行查看。源码下载地址在文章末尾。】

翻译 by 明明如月 QQ 605283073】

原文地址：http://websystique.com/springmvc/spring-mvc-4-restful-web-services-crud-example-resttemplate/

上一篇：

Spring 4 MVC @RestController 注解实现REST Service

下一篇：Spring MVC 4 文件上传下载 Hibernate+MySQL例子 （带源码）

本文非常好，推荐大家好好看看，很多人理解的restful不对

本文我们将使用Spring

MVC 4实现 CRUD Restful WebService , 通过RestTemplate写一个 REST 客户端，定义这些服务. 我们也可以通过外部的一些客户端来测试这些服务。

简短 & 快速介绍REST

REST表示 Representational State Transfer（表示性状态转换）.

它是可以用来设计web services的框架，可以被不同的客户端调用。

核心思想是：使用简单的HTTP协议来实现调用，而不是CORBA, RPC

或者 SOAP等负责的机制。

在Rest 基础设计中，资源使用以下动词进行操作。

创建资源 : 使用 HTTP POST

获取资源 : 使用 HTTP GET

更新资源 : 使用 HTTP PUT

删除资源 : 使用 HTTP DELETE

也意味着,你作为Rest 服务开发者或者客户，应该遵循以上的标准。

尽管没有限制必须返回的类型，但是一般基于Web services的Rest返回JSON或者XML作为响应。

客户端可以指定（使用HTTP Accept header）他们想要的资源类型吗，服务器返回需要的资源。

指明资源的Content-Type。如果想详细的理解

restful可以参考这里：StackOverflow

link

基于Rest的Controller（控制器）

我们的 REST API :

GET 方式请求 /api/user/ 返回用户列表

GET 方式请求 /api/user/1返回id为1的用户

POST 方式请求 /api/user/ 通过user对象的JSON 参数创建新的user对象

PUT 方式请求 /api/user/3 更新id为3的发送json格式的用户对象

DELETE 方式请求/api/user/4删除 ID为 4的user对象

DELETE 方式请求/api/user/删除所有user

```
[java]
```

```java
package com.websystique.springmvc.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.util.UriComponentsBuilder;

import com.websystique.springmvc.model.User;
import com.websystique.springmvc.service.UserService;

@RestController
public class HelloWorldRestController {

    @Autowired
    UserService userService;  //Service which will do all data retrieval/manipulation work


    //-------------------Retrieve All Users--------------------------------------------------------

    @RequestMapping(value = "/user/", method = RequestMethod.GET)
    public ResponseEntity<List<User>> listAllUsers() {
        List<User> users = userService.findAllUsers();
        if(users.isEmpty()){
            return new ResponseEntity<List<User>>
(HttpStatus.NO_CONTENT);//You many decide to return HttpStatus.NOT_FOUND
        }
        return new ResponseEntity<List<User>>(users, HttpStatus.OK);
    }


    //-------------------Retrieve Single User--------------------------------------------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET, produces = MediaType.APPLICATION_J
    public ResponseEntity<User> getUser(@PathVariable("id") long id) {
        System.out.println("Fetching User with id " + id);
        User user = userService.findById(id);
        if (user == null) {
            System.out.println("User with id " + id + " not found");
            return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<User>(user, HttpStatus.OK);
    }



    //-------------------Create a User--------------------------------------------------------

    @RequestMapping(value = "/user/", method = RequestMethod.POST)
    public ResponseEntity<Void> createUser(@RequestBody User user,    UriComponentsBuilder ucBuilder) {
        System.out.println("Creating User " + user.getName());

        if (userService.isUserExist(user)) {
            System.out.println("A User with name " + user.getName() + " already exist");
            return new ResponseEntity<Void>(HttpStatus.CONFLICT);
        }

        userService.saveUser(user);

        HttpHeaders headers = new HttpHeaders();
        headers.setLocation(ucBuilder.path("/user/{id}").buildAndExpand(user.getId()).toUri());
        return new ResponseEntity<Void>(headers, HttpStatus.CREATED);
    }


    //------------------- Update a User --------------------------------------------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.PUT)
    public ResponseEntity<User> updateUser(@PathVariable("id") long id, @RequestBody User user) {
        System.out.println("Updating User " + id);

        User currentUser = userService.findById(id);

```

```
        if (currentUser==null) {
            System.out.println("User with id " + id + " not found");
            return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
        }

        currentUser.setName(user.getName());
        currentUser.setAge(user.getAge());
        currentUser.setSalary(user.getSalary());

        userService.updateUser(currentUser);
        return new ResponseEntity<User>(currentUser, HttpStatus.OK);
    }

    //------------------- Delete a User --------------------------------------------------------

    @RequestMapping(value = "/user/{id}", method = RequestMethod.DELETE)
    public ResponseEntity<User> deleteUser(@PathVariable("id") long id) {
        System.out.println("Fetching & Deleting User with id " + id);

        User user = userService.findById(id);
        if (user == null) {
            System.out.println("Unable to delete. User with id " + id + " not found");
            return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
        }

        userService.deleteUserById(id);
        return new ResponseEntity<User>(HttpStatus.NO_CONTENT);
    }


    //------------------- Delete All Users --------------------------------------------------------

    @RequestMapping(value = "/user/", method = RequestMethod.DELETE)
    public ResponseEntity<User> deleteAllUsers() {
        System.out.println("Deleting All Users");

        userService.deleteAllUsers();
        return new ResponseEntity<User>(HttpStatus.NO_CONTENT);
    }

}
```

详解：
@RestController :首先我们使用的是Spring 4的新注解 @RestController注解.

此注解避免了每个方法都要加上@ResponseBody注解。也就是说@RestController
自己戴上了 @ResponseBody注解,看以看作是

@Controller
和 @ResponseBody的结合体。

@RequestBody : 如果方法参数被 @RequestBody注解，Spring将绑定HTTP请求体到那个参数上。如果那样做，Spring将根据请求中的ACCEPT或者 Content-Type
header（私下）使用 HTTP
Message converters 来将http请求体转化为domain对象。

@ResponseBody : 如果方法加上了@ResponseBody注解，Spring返回值到响应体。如果这样做的话，Spring将根据请求中的 Content-Type
header（私下）使用 HTTP
Message converters 来将domain对象转换为响应体。

ResponseEntity 是一个真实数据.它代表了整个 HTTP 响应（response）. 它的好处是你可以控制任何对象放到它内部。

你可以指定状态码、头信息和响应体。它包含你想要构建HTTP Response 的信息。

@PathVariable 此注解意味着一个方法参数应该绑定到一个url模板变量[在'{}'里的一个]中

一般来说你，要实现REST API in Spring 4 需要了解@RestController , @RequestBody, ResponseEntity 和 @PathVariable 这些注解 .另外, spring 也提供了一些支持类帮助你实现一些可定制化的东西。

MediaType : 带着 @RequestMapping 注解,通过特殊的控制器方法你可以额外指定,MediaType来生产或者消耗。

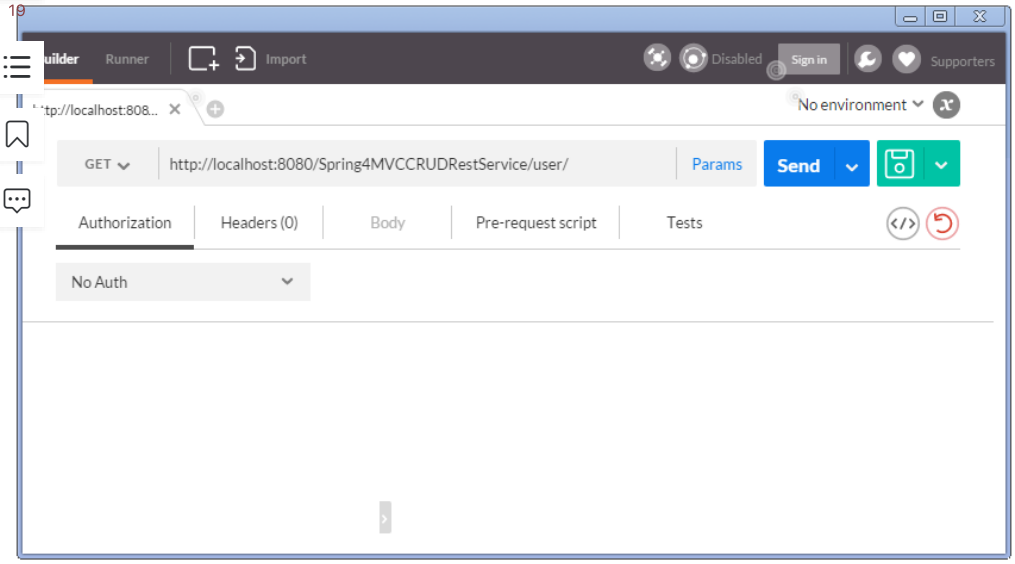
发布和测试此API

http://localhost:8080/Spring4MVCCRUDRestService.

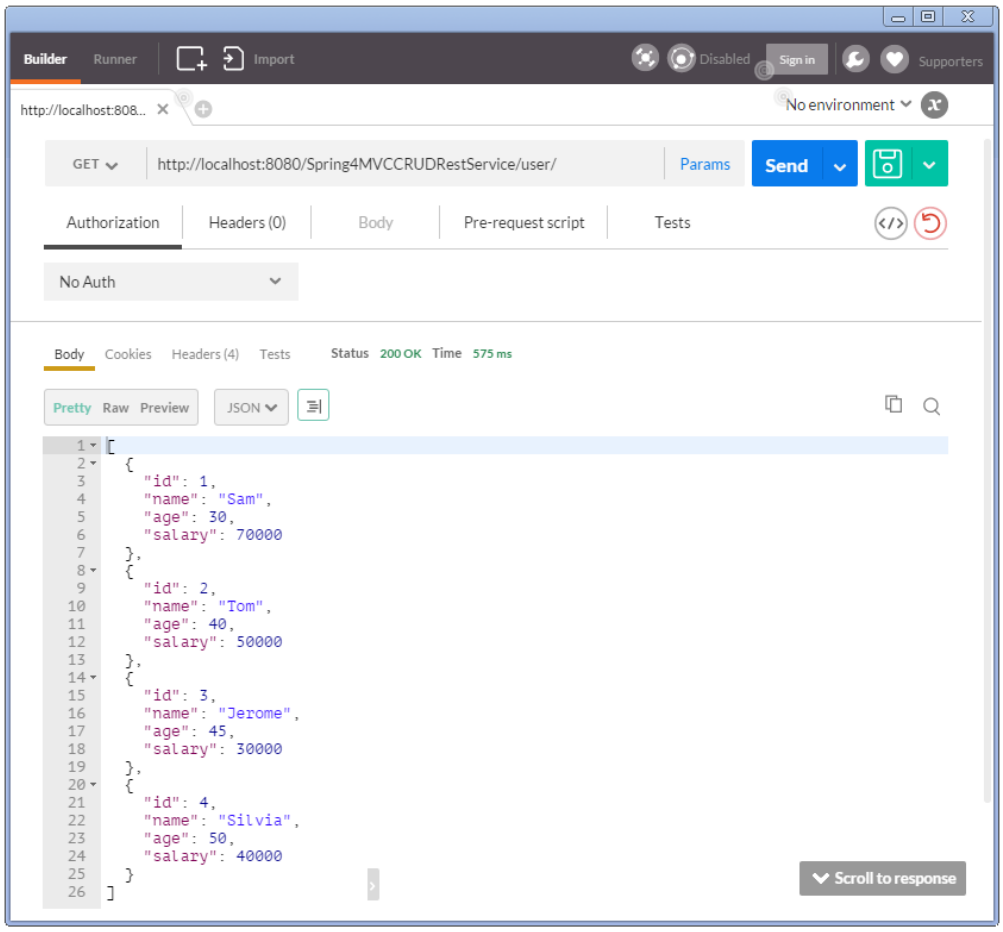
想要测试此API，我将使用POSTMAN这个外部客户端，接下来我们也将写我们自己的客户端。

1. 获取所有用户

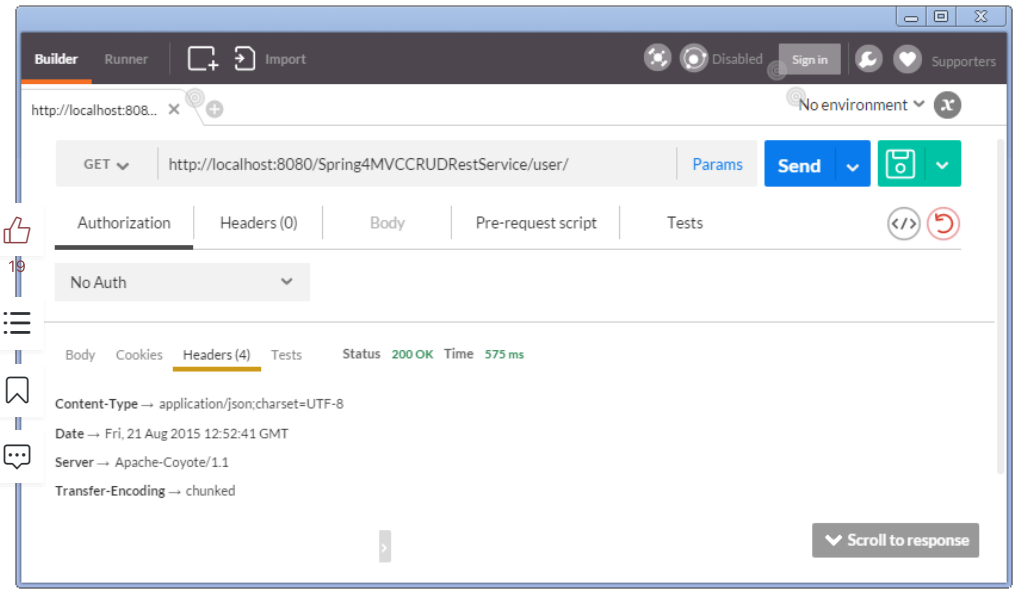
使用 POSTMAN工具，选择请求类型为GET，指明uri



注意：我们没有指明任何HTTP头。点击 发送，将接收到所有用户的列表



也要注意HTTP 200 响应。



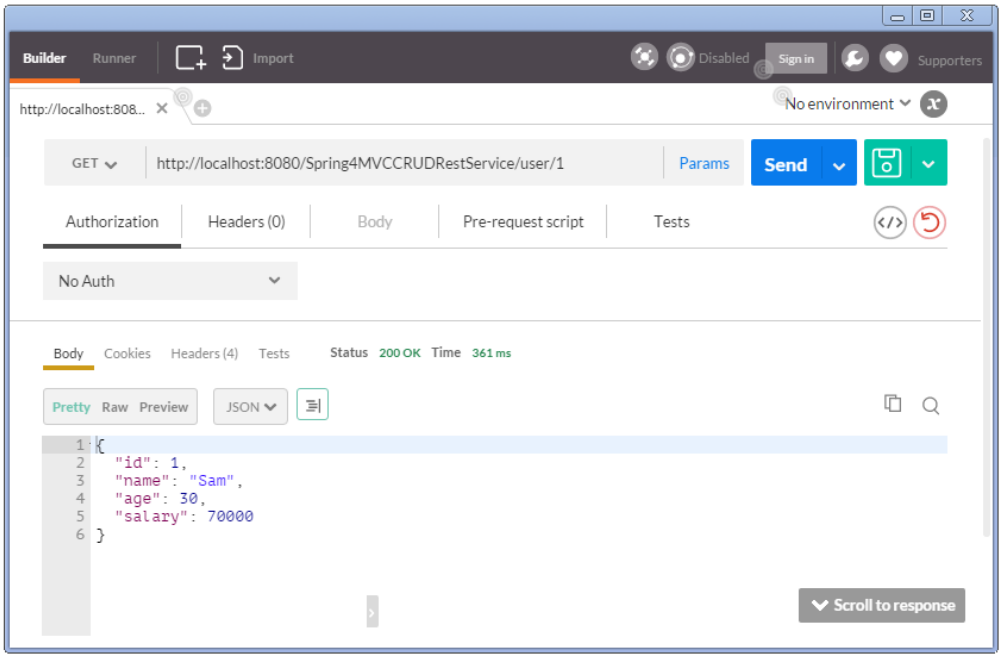
你也许好奇为什么此响应通过JSON字符串发送的，在响应里的Content-Type 头说明了这个。
因为我们添加了JACKSON

```
[html]

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.5.3</version>
</dependency
```

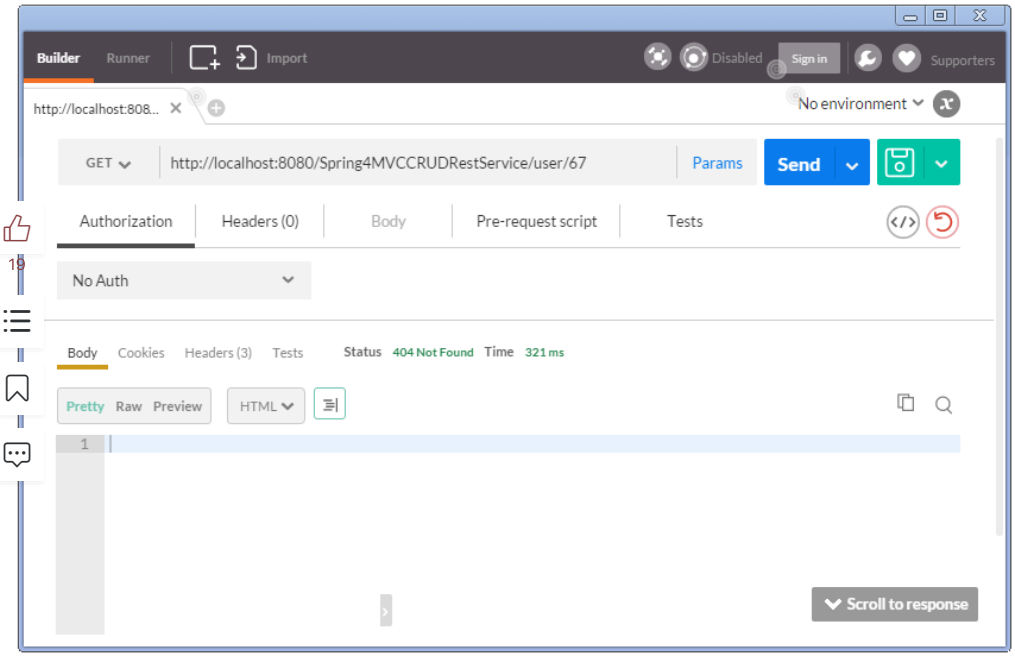
因为Spring在类路径发现了这个库，它调用了内置的MappingJackson2HttpMessageConverter 转换器将响应（对象集合）转换为JSON格式。
Spring内置转换器的好处是，大部分情况下只要把库放到类路径，即可完成转换。当然了有时候我们也需要采用我们的API。比如，如果我们像也提供XML格式的话，我们需要对User类加上JAXB注解。

2. 获取单个用户

GET方式 指定/user/1

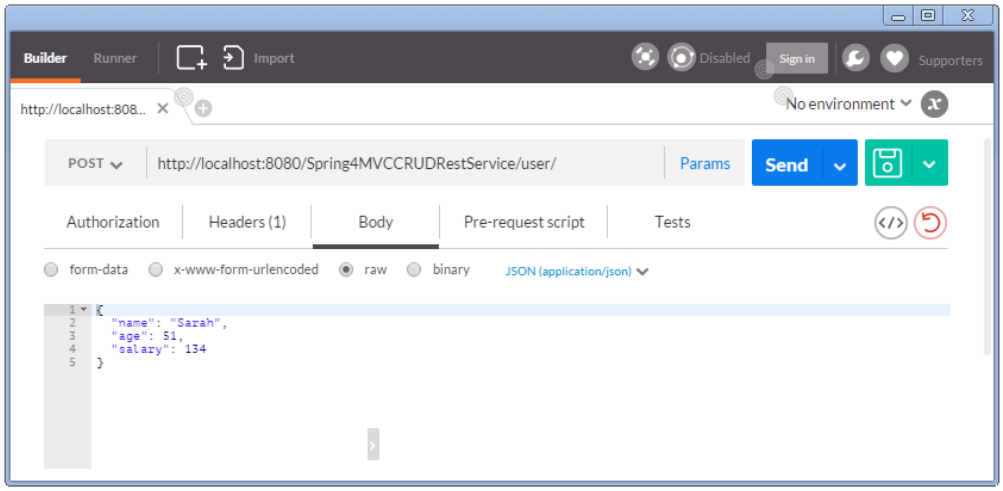


现在试着发送一个带有错误识别码的GET请求，将收到一个HTTP 404

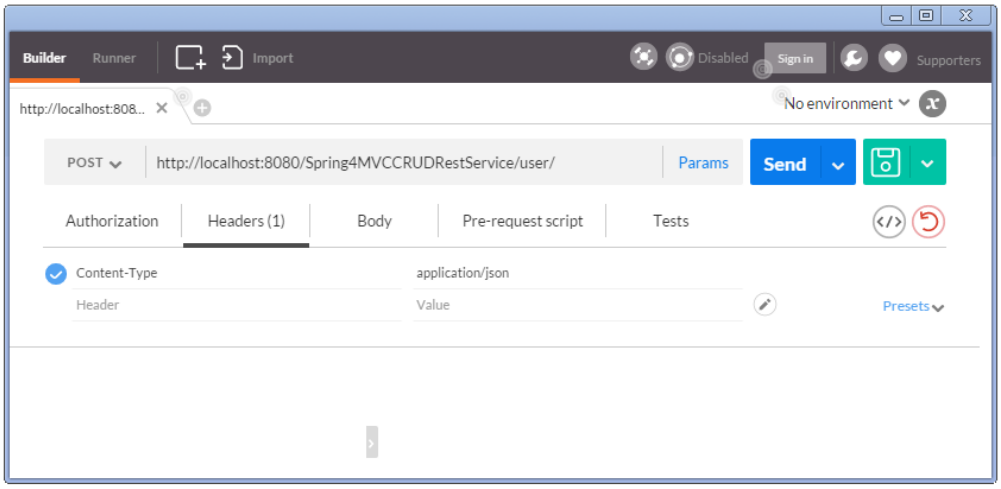


3.创建一个 User

选择POST方法，指明uri /user/ 指明POSTMAN Body选项卡，选择application/json类型

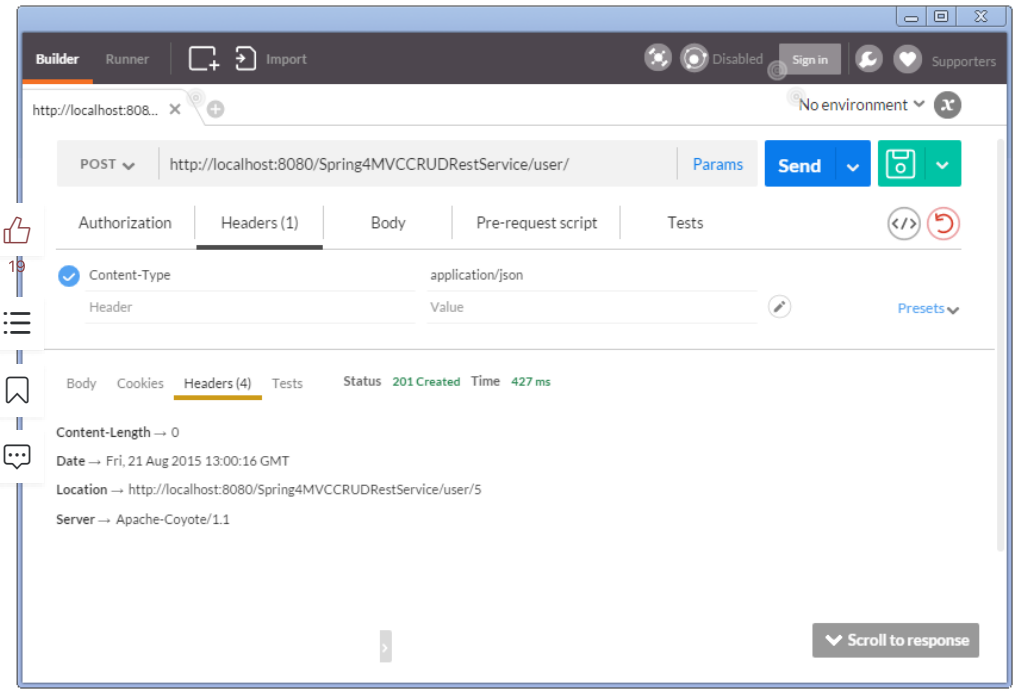


你要注意POSTMAN自动添加了Content-Type 头信息

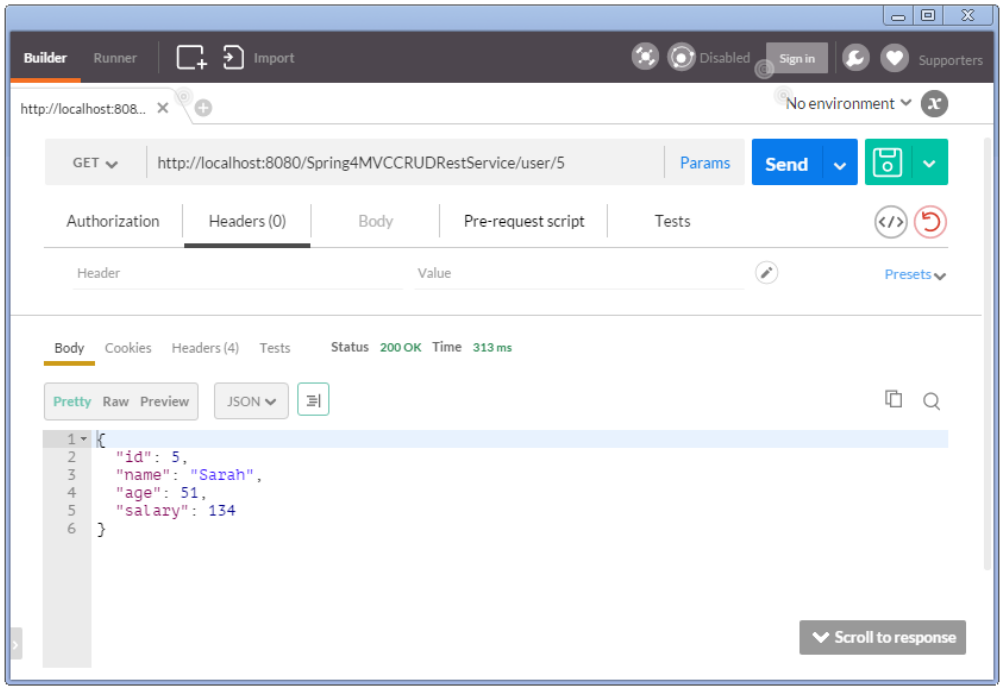


记住： Accept header包含client能给识别的类型。 Content-Type header表示数据的实际类型。

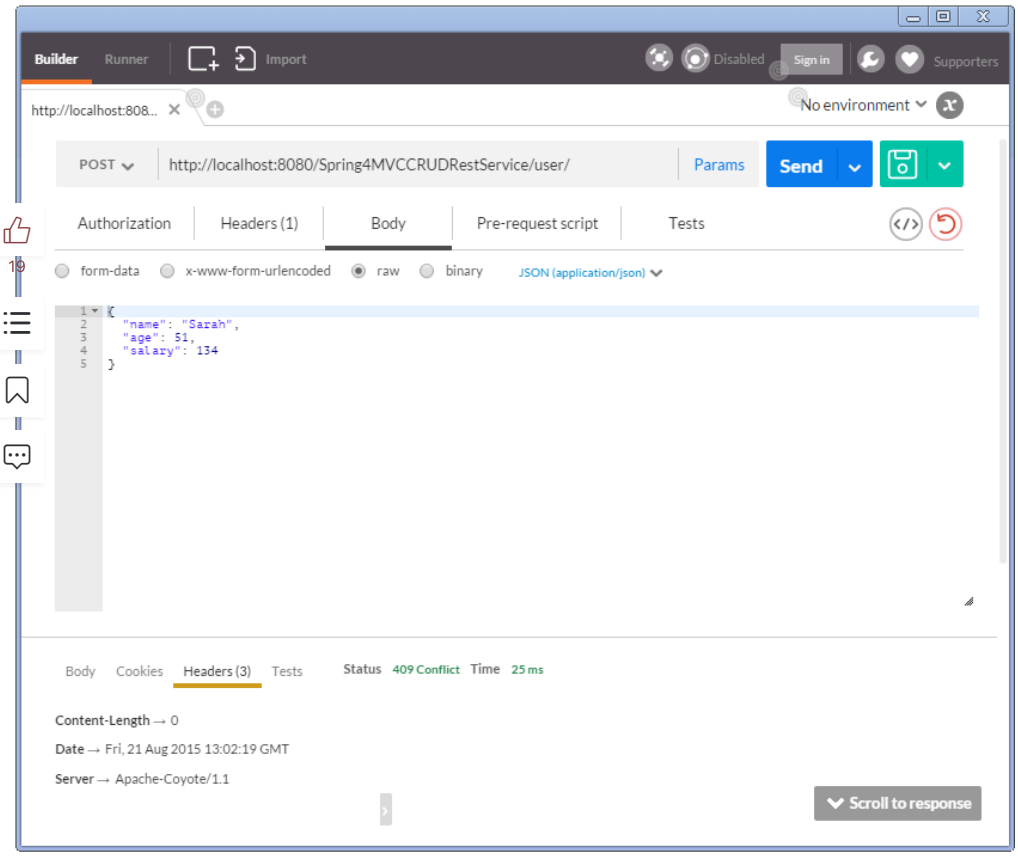
点击发送以后 将收到 HTTP 200 没有响应体（api里面没有在响应体发送任何东西）



你可以查询新创建的用户

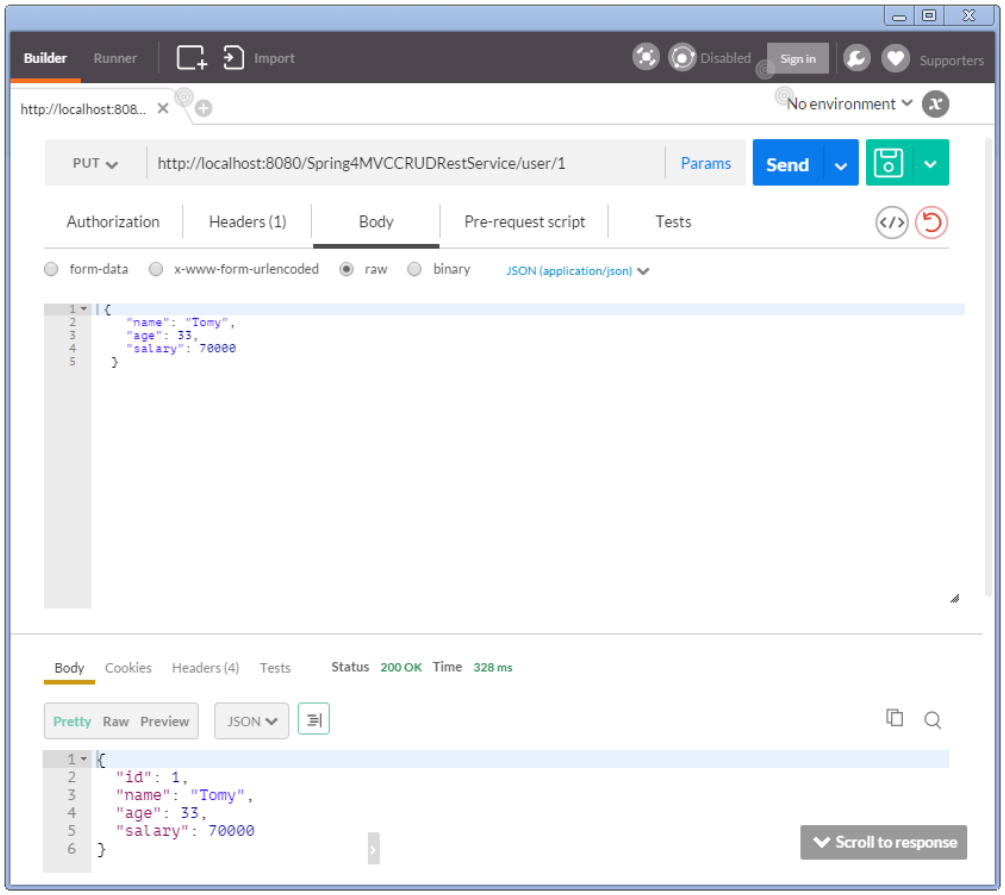


这是实现REST的普通实现方式。但是也没人阻止你为POST或者PUT方式响应体里发送内容。但是这还是REST的API？值得怀疑。
不管怎样，我们试着创建同一个用户时，你将获得HTTP冲突的响应。



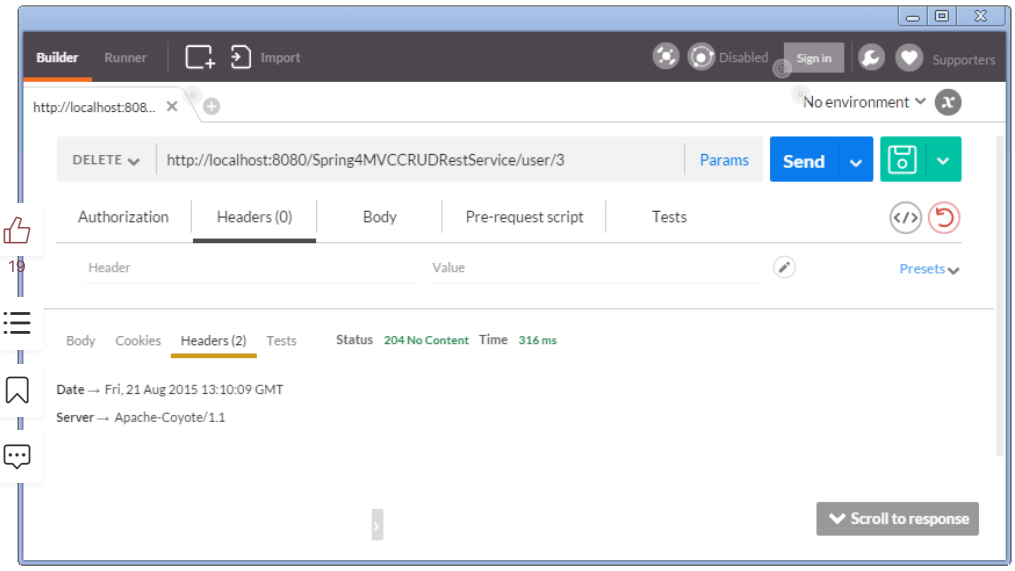
4.更新用户

发送一个HTTP PUT 请求来更新用户。

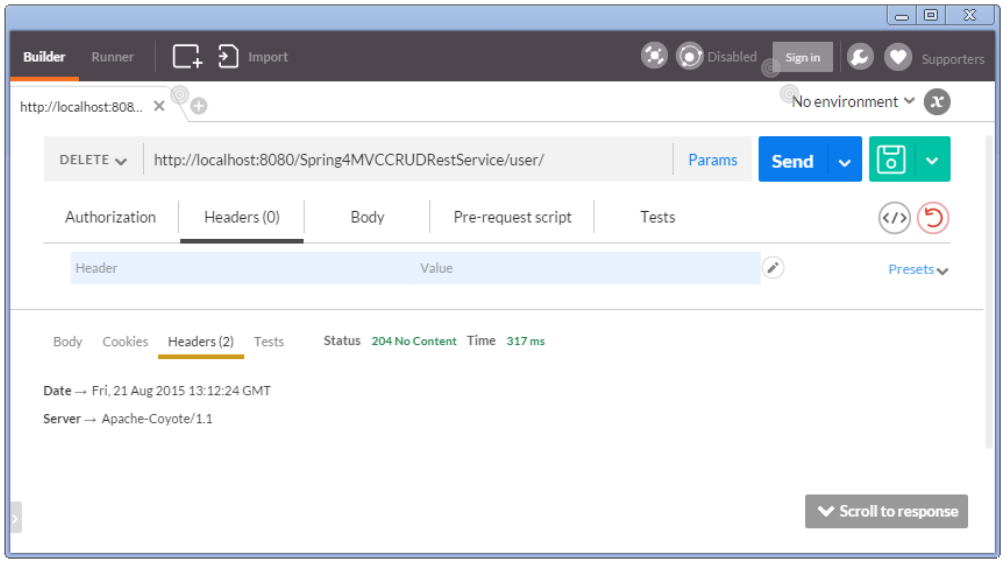


注意：这次我们接收到了响应体。这是因为在控制器的方法实现里我们发送了数据。再次强调，有的人也许不在响应体里面发送更新的详情，只发送位置头（和创建用户一样）。

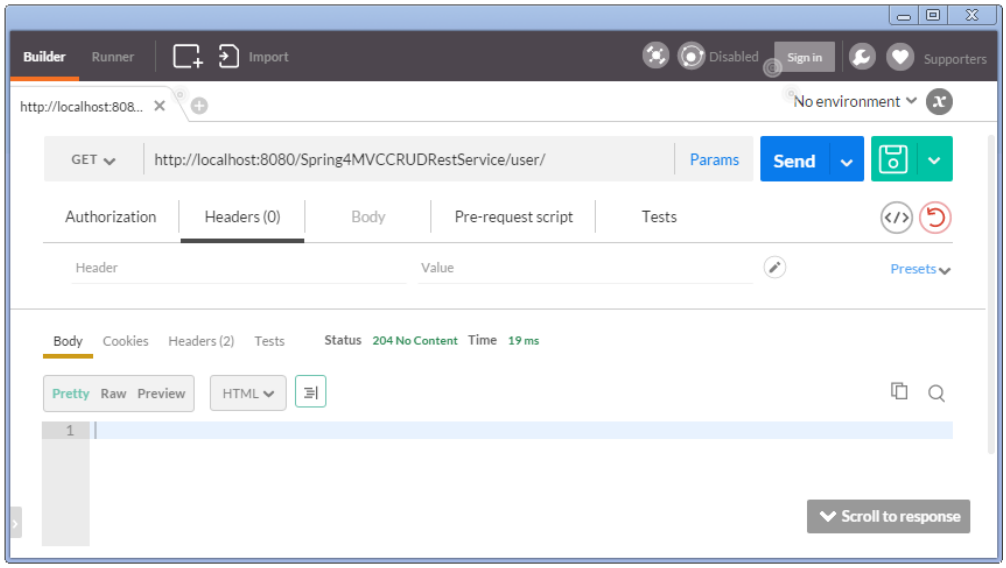
5.删除用户



6 删除所有用户



7.删除用户后验证



根据RestTemplate 写REST Client

Postman是测试Rest Api的超好用的工具，但是如果你想完整的消化REST，可以尝试自己写一个。
最出名的Htpp 客户端是HttpClient（Apache HttpComponents）。

但是用它来访问REST service则相对少见。
Spring的 RestTemplate随之出现。RestTemplate 提供了高级方法，来响应者6种主要的HTTP方法。

HTTP 方法和对应的 RestTemplate方法:


19

HTTP GET : getForObject, getForEntity

HTTP PUT : put(String url, Object request, String…urlVariables)

HTTP DELETE : delete

HTTP POST : postForLocation(String url, Object request, String… urlVariables), postForObject(String url, Object request, ClassresponseType, String… uriVariables)

HTTP HEAD : headForHeaders(String url, String… urlVariables)

HTTP OPTIONS : optionsForAllow(String url, String… urlVariables)

HTTP PATCH and others : exchange execute

定义 Rest client , 定义REST services

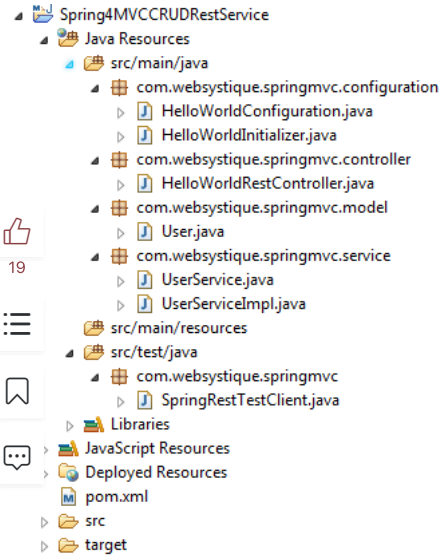
```java
package com.websystique.springmvc;

import java.net.URI;
import java.util.LinkedHashMap;
import java.util.List;

import org.springframework.web.client.RestTemplate;

import com.websystique.springmvc.model.User;

public class SpringRestTestClient {

    public static final String REST_SERVICE_URI = "http://localhost:8080/Spring4MVCCRUDRestService";

    /* GET */
    @SuppressWarnings("unchecked")
    private static void listAllUsers(){
        System.out.println("Testing listAllUsers API-----------");

        RestTemplate restTemplate = new RestTemplate();
        List<LinkedHashMap<String, Object>> usersMap = restTemplate.getForObject(REST_SERVICE_URI+"/user

        if(usersMap!=null){
            for(LinkedHashMap<String, Object> map : usersMap){
                System.out.println("User : id="+map.get("id")+", Name="+map.get("name")+", Age="+map.get
            }
        }else{
            System.out.println("No user exist----------");
        }
    }

    /* GET */
    private static void getUser(){
        System.out.println("Testing getUser API----------");
        RestTemplate restTemplate = new RestTemplate();
        User user = restTemplate.getForObject(REST_SERVICE_URI+"/user/1", User.class);
        System.out.println(user);
    }

    /* POST */
    private static void createUser() {
        System.out.println("Testing create User API----------");
        RestTemplate restTemplate = new RestTemplate();
        User user = new User(0,"Sarah",51,134);
        URI uri = restTemplate.postForLocation(REST_SERVICE_URI+"/user/", user, User.class);
        System.out.println("Location : "+uri.toASCIIString());
    }

    /* PUT */
    private static void updateUser() {
        System.out.println("Testing update User API----------");
        RestTemplate restTemplate = new RestTemplate();
        User user  = new User(1,"Tomy",33, 70000);
        restTemplate.put(REST_SERVICE_URI+"/user/1", user);
        System.out.println(user);
```

```
    }

    /* DELETE */
    private static void deleteUser() {
        System.out.println("Testing delete User API----------");
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.delete(REST_SERVICE_URI+"/user/3");
    }


    /* DELETE */
    private static void deleteAllUsers() {
        System.out.println("Testing all delete Users API----------");
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.delete(REST_SERVICE_URI+"/user/");
    }

    public static void main(String args[]){
        listAllUsers();
        getUser();
        createUser();
        listAllUsers();
        updateUser();
        listAllUsers();
        deleteUser();
        listAllUsers();
        deleteAllUsers();
        listAllUsers();
    }
}
```

重启服务器，运行上面的程序。
下面是输出：

```
Testing listAllUsers API-----------
User : id=1, Name=Sam, Age=30, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
Testing getUser API----------
User [id=1, name=Sam, age=30, salary=70000.0]
Testing create User API----------
Location : http://localhost:8080/Spring4MVCCRUDRestService/user/5
Testing listAllUsers API-----------
User : id=1, Name=Sam, Age=30, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0
Testing update User API----------
User [id=1, name=Tomy, age=33, salary=70000.0]
Testing listAllUsers API-----------
User : id=1, Name=Tomy, Age=33, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=3, Name=Jerome, Age=45, Salary=30000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0
Testing delete User API----------
Testing listAllUsers API-----------
User : id=1, Name=Tomy, Age=33, Salary=70000.0
User : id=2, Name=Tom, Age=40, Salary=50000.0
User : id=4, Name=Silvia, Age=50, Salary=40000.0
User : id=5, Name=Sarah, Age=51, Salary=134.0
Testing all delete Users API----------
Testing listAllUsers API-----------
No user exist----------
```

完整的例子



更新pom.xml添加项目依赖

```
[html]
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.websystique.springmvc</groupId>
  <artifactId>Spring4MVCCRUDRestService</artifactId>
  <packaging>war</packaging>
  <version>1.0.0</version>
  <name>Spring4MVCCRUDRestService Maven Webapp</name>

    <properties>
        <springframework.version>4.2.0.RELEASE</springframework.version>
        <jackson.version>2.5.3</jackson.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${springframework.version}</version>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson.version}</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
        </dependency>

    </dependencies>


    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.2</version>
                    <configuration>
                        <source>1.7</source>
                        <target>1.7</target>
                    </configuration>
                </plugin>
```

```
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>2.4</version>
                <configuration>
                    <warSourceDirectory>src/main/webapp</warSourceDirectory>
                    <warName>Spring4MVCCRUDRestService</warName>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
        </plugins>
    </pluginManagement>

    <finalName>Spring4MVCCRUDRestService</finalName>
  </build>
</project>
```

er Service

```java
package com.websystique.springmvc.service;

import java.util.List;

import com.websystique.springmvc.model.User;



public interface UserService {

    User findById(long id);

    User findByName(String name);

    void saveUser(User user);

    void updateUser(User user);

    void deleteUserById(long id);

    List<User> findAllUsers();

    void deleteAllUsers();

    public boolean isUserExist(User user);

}
```

```java
package com.websystique.springmvc.service;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.concurrent.atomic.AtomicLong;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.websystique.springmvc.model.User;

@Service("userService")
@Transactional
public class UserServiceImpl implements UserService{

    private static final AtomicLong counter = new AtomicLong();

    private static List<User> users;

    static{
        users= populateDummyUsers();
    }

    public List<User> findAllUsers() {
        return users;
```

```
    }

    public User findById(long id) {
        for(User user : users){
            if(user.getId() == id){
                return user;
            }
        }
        return null;
    }

    public User findByName(String name) {
        for(User user : users){
            if(user.getName().equalsIgnoreCase(name)){
                return user;
            }
        }
        return null;
    }

    public void saveUser(User user) {
        user.setId(counter.incrementAndGet());
        users.add(user);
    }

    public void updateUser(User user) {
        int index = users.indexOf(user);
        users.set(index, user);
    }

    public void deleteUserById(long id) {

        for (Iterator<User> iterator = users.iterator(); iterator.hasNext(); ) {
            User user = iterator.next();
            if (user.getId() == id) {
                iterator.remove();
            }
        }
    }

    public boolean isUserExist(User user) {
        return findByName(user.getName())!=null;
    }

    private static List<User> populateDummyUsers(){
        List<User> users = new ArrayList<User>();
        users.add(new User(counter.incrementAndGet(),"Sam",30, 70000));
        users.add(new User(counter.incrementAndGet(),"Tom",40, 50000));
        users.add(new User(counter.incrementAndGet(),"Jerome",45, 30000));
        users.add(new User(counter.incrementAndGet(),"Silvia",50, 40000));
        return users;
    }

    public void deleteAllUsers() {
        users.clear();
    }

}
```

Model （模型）类

```java
package com.websystique.springmvc.model;

public class User {

    private long id;

    private String name;

    private int age;

    private double salary;

    public User(){
        id=0;
    }

    public User(long id, String name, int age, double salary){
```

```
        this.id = id;
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + (int) (id ^ (id >>> 32));
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        User other = (User) obj;
        if (id != other.id)
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "User [id=" + id + ", name=" + name + ", age=" + age
                + ", salary=" + salary + "]";
    }


}
```

配置类

```
[java]
package com.websystique.springmvc.configuration;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;

```

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.websystique.springmvc")
public class HelloWorldConfiguration {


}
```



```java
package com.websystique.springmvc.configuration;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class HelloWorldInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { HelloWorldConfiguration.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

}
```

为你的REST API添加CORS支持

当访问REST API时，你可能需要面对“同源策略”问题。
错误如下：
” No ‘Access-Control-Allow-Origin’ header is present on the requested resource. Origin ‘http://127.0.0.1:8080′ is therefore not allowed access.” OR
” XMLHttpRequest cannot load http://abc.com/bla. Origin http://localhost:12345 is not allowed by Access-Control-Allow-Origin.”

一般来说，在服务器端，我们在响应中返回额外的CORS访问控制头，实现跨域链接。
用 Spring的话，我么可以写一个简单的过滤器为每个响应添加CORS特征头。

```java
package com.websystique.springmvc.configuration;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletResponse;


public class CORSFilter implements Filter {

    public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOExcepti
        System.out.println("Filtering on...........................................................");
        HttpServletResponse response = (HttpServletResponse) res;
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "POST, GET, PUT, OPTIONS, DELETE");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers", "x-requested-with");
        chain.doFilter(req, res);
    }

    public void init(FilterConfig filterConfig) {}
```

```
    public void destroy() {}

}
```

需要将其添加在Spring 配置中：

```java
package com.websystique.springmvc.configuration;

import javax.servlet.Filter;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class HelloWorldInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return new Class[] { HelloWorldConfiguration.class };
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return null;
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }

    @Override
    protected Filter[] getServletFilters() {
        Filter [] singleton = { new CORSFilter()};
        return singleton;
    }

}
```

源码下载：http://websystique.com/?smd_process_download=1&download_id=1689
源码（带SORS）下载：http://websystique.com/?smd_process_download=1&download_id=1890

特别说明：此系列教程有的童鞋下载下来运行 经常404 或者改成xml方式以后
缺少org.springframework.web.context.ContextLoaderServlet等
参见：http://blog.csdn.net/w605283073/article/details/52126347

目前您尚未登录，请 登录 或 注册 后进行评论

hugechun 2017-12-07 10:26 1条回复 回复 14楼
很厉害啊

yuanyeguishu 2017-12-05 19:49 回复 13楼
大佬很给力，很rest式， 膜拜…… 感谢……

macknight 2017-11-02 08:51 回复 12楼
请问 http://localhost:8080/Spring4MVCCRUDRestService
中的 /Spring4MVCCRUDRestService是在哪里设置的?